Attorney Reference: 19378.0023

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

Johan Ageby et al.

Application. No.: 10/025,991

Art Unit: 2661

Filed: December 26, 2001

Title: METHODS FOR CONTROLLING RESOURCES IN A COMMUNICATION NETWORK

CLAIM FOR PRIORITY

Assistant Commissioner for Patents
Washington, DC 20231

Sir:

Certified copies of corresponding Swedish Application No. 0004896-7, filed December 29, 2000, and Swedish Application No. 0102471-0, filed July 10, 2001, are attached. It is requested that the right of priority provided by 35 U.S.C. 119 be extended by the Patent Office.

Respectfully submitted,

Date: March 20, 2002

Eric J. Franklin

Eric J. Franklin, Reg. No. 37,134
Swidler Berlin Shereff Friedman, LLP
3000 K Street, NW, Suite 300
Washington, DC 20007
Telephone: (202) 424-7500

PRV
PATENT- OCH REGISTRERINGSVERKET
Patentavdelningen

OIPE JC76
MAR 20 2002
PATENT & TRADEMARK OFFICE

RECEIVED
MAR 2 2 2002
Technology Center 2600

REGISTRERINGSVERKET • PATENT- OCH

Intyg
Certificate

Härmed intygas att bifogade kopior överensstämmer med de handlingar som ursprungligen ingivits till Patent- och registreringsverket i nedannämnda ansökan.

This is to certify that the annexed is a true copy of the documents as originally filed with the Patent- and Registration Office in connection with the following patent application.

(71) Sökande Applicant (s) *Net Insight AB, Stockholm SE*

(21) Patentansökningsnummer Patent application number *0102471-0*

(86) Ingivningsdatum Date of filing *2001-07-10*

CERTIFIED COPY OF PRIORITY DOCUMENT

Stockholm, 2002-01-02

För Patent- och registreringsverket
For the Patent- and Registration Office

Christina Vängborg

Christina Vängborg

Avgift
Fee 170:-

PATENT- OCH REGISTRERINGSVERKET
SWEDEN

Postadress/Adress
Box 5055
S-102 42 STOCKHOLM

Telefon/Phone
+46 8 782 25 00
Vx 08-782 25 00

Telex
17978
PATOREG S

Telefax
+46 8 666 02 86
08-666 02 86

jw/

Ref.: 55324 SE

Applicant: Net Insight AB

Dynamic Protocol

THE BACKGROUND OF THE INVENTION AND PRIOR ART

The present invention relates generally to a telecommunications protocol, which provides a mechanism for negotiating resources between interfaces in the system. More particularly the invention relates to a method of allocating resources in a synchronous time division multiplex communications system according to the preamble of claim 1 and a communications system according to the preamble of claim 11. The invention also relates to a computer program according to the preamble of claim 9 and a computer readable medium according to claim 10.

The known protocols for allocating resources in communications system where the transmission resources are constituted by time slots in a repeating frame structure have presumed that the resources are allocated at initial configuration of the system. Such procedure requires substantial planning by the network operator and is very inflexible to later alterations of the system's topology and/or changes in the resource requirement from the various interfaces in the system.

Furthermore, there is a risk that the system is configured such that an overlap may occurs, between the configured resources. For instance, the number of configured resources may not correspond to the number of actually available resources.

SUMMARY OF THE INVENTION

The object of the present invention is therefore to provide a resource allocating solution, which alleviates the problems above and thus offers a simple and adaptable distribution of resources in a system of any size.

According to another aspect of the invention the object is achieved by a method of allocating resources in a synchronous time division multiplex communications system, as initially described, which is characterized by the following steps; sending a link status message from an interface whenever the interface registers a change in the topology of the system, sending a gather message from an interface whenever the interface requests a revision of a current ownership distribution of resources, sending a sync message from the master interface as an indication of a current distribution of ownership with respect to the resources between the interfaces in the system, and, for each interface, generating a distribution of the ownership to the resources on basis of the interface's topological position and a latest received sync message.

According to a further aspect of the invention the object is achieved by a computer program directly loadable into the internal memory of a computer, comprising software for performing the above proposed method when said program is run on a computer.

According to another aspect of the invention the object is achieved by a computer readable medium, having a program recorded thereon, where the program is to make a computer perform the proposed method.

According to one aspect of the invention the object is achieved by a communications system as initially described, which is characterized in that it comprises at least one node, which in turn includes one or more of the interfaces. The node is presumed to be adapted to effect the proposed method.

The invention offers an efficient, reliable and fair solution for dynamically allocating transmission resources in a communications system. The proposed solution on one hand makes manual configuration unnecessary. On the other hand, if a system still is manually configured, the invention safeguards against any erroneous or conflicting configurations. This is, of course a very desirable feature from a network operator's point-of-view.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is now to be explained more closely by means of preferred embodiments, which are disclosed as examples, and with reference to the attached drawings.

Figures 1a-d show different examples of allocation domains to which the invention is applicable,

Figure 2 illustrates a so-called short-circuit scenario,

Figure 3 illustrates a typical probe according to an embodiment of the invention,

Figure 4 illustrates a typical borrow and return of resources according to an embodiment of the invention,

Figure 5 shows important transitions of a quark machine according to an embodiment of the invention,

Figure 6 shows an example of messages sent when an interface is appended to a bus according to an embodiment of the invention,

Figure 7 shows a first example of messages sent when a ring is closed according to an embodiment of the invention,

Figure 8 shows a second example of messages sent when a ring is closed according to an embodiment of

the invention, and

Figures 9a, b show two important cases in which resources are allocated according to embodiments of the invention.

DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

This proposed solution relates to a mechanisms for negotiating resources between interfaces in a synchronous time division multiplex communications system having a master interface communicating with one or more slave interfaces, and in which the resources between the interfaces are represented by time slots in a repeating frame structure. Thus, the solution may be applied in a system of dynamic synchronous transfer mode (DTM) type. In such system, the solution can be accomplished by the Resource Management Protocol (DRMP). DRMP is a token passing mechanism for negotiating between interfaces which resource units that are available. It is divided into two orthogonal mechanisms, ownership and access.

Definitions

allocation domain: the same as a bypass chain but if the topology is point-to-point or bus, the last node is not counted as member of the AD.

access right: the right to write on a certain slot on an interface for a certain number of bypass hops

access token: a token that is passed around to grant access right between interfaces on an allocation domain

bool: a logical variable type, which can take two values, true or false

fairness algorithm: the algorithm used to determine actual ownership ranges from a set of requested policies from the interfaces in the AD

master interface: the interface in the AD having the lowest mac address.

ownership: the obligation to manage an access token with respect to lending and issuing probe and kill messages
quark: the smallest resource unit it is one slot wide and one bypasshop long
topology: a set of two or more interfaces connected in a bypass chain that is either closed or open

Abbreviations

For the purpose of the present document, the following abbreviations apply:

AD	Allocation Domain
BR	BitRate
DCC	DTM Control Channel
Distown	Distribute Ownership
DLC	Data Link Change
DLSP	DTM Link State Protocol
DRMP	DTM Resource Management Protocol
DO	Dynamic Ownership
Mac	MAC Address.
Msg	Message
Prrpy	probe reply
Qreq	quark request
Qret	quark return
Qt	quark transfer
Ptp	Point-to-point

Problem domain

DRMP efficiently distributes write access rights to the time slots in a DTM system. This is done with the following mechanisms:

- Ownership and Announcement of resources.
- Borrowing and Probing of resource tokens.

Resource ownership

Consider the everyday concept of ownership. Ownership does not automatically grant access right. Remember that even if something is owned by a certain unit, some other unit might have borrowed it and thus made it unaccessible to the owner.

Static ownership

In this mode, all the resource units have had their ownerships defined at startup time. It is still possible to change the ownership distribution of a running system. This requires manual intervention by the operator, often in more than one unit, something that is very complicated.

Dynamic ownership

In this mode, ownership is negotiated by letting all units initially state how many resources they want. The master interface then distributes these policies to all the participating interfaces, including the master itself. Dynamic mode eliminates the necessity for configuration by the operator. Overlapping may occur transiently but the DO ensures that this state does not persist and that it is never dangerous.

Resource announcement

Resource announcement is done by each unit telling all the other units how many tokens it is willing to lend out. When a unit receives an announcement it stores that information. This is used when borrowing.

Later it uses this information to decide which units to borrow from.

Borrowing

If a unit finds that it has not got enough resources it will attempt to borrow access rights to the other units resources. This is done by passing tokens between the borrower and the most suitable owners according to the announce tables. When the tokens are to be returned they are always returned to the owner.

Probe / Kill

The Probe mechanism has two main objectives:

- Recreate lost resource tokens or initially create them.
- Resolve situations when two units simultaneously claim to have the access right to a resource.

The probing is carried out in the same way for both cases. In order to do a successful probe of a set of tokens, the owner of the resource must make one query to all other units sharing this

token and get a reply from all of them. There are three main outcomes of a successfull probe. The token may be non-allocated, allocated once or doubly allocated. Probing involves sending many messages across the network, especially at bootstrap. This can take considerable time. A useful special case is the point-to-point topology, in which the two units create the resources locally.

Fault handling

The fault situations and what DRMP does to fix them when they have occurred are described below.

Borrowing failed

Since the number of resources announced by a unit only is valid at the time of the announcement, borrowing does not always succeed. When a unit has attempted to borrow tokens and have not managed to find the minimum required.

Ownership overlapping

Ownership overlapping causes a problem in the static ownership mode only. The dynamic ownership protocol ensures that either the ownerships are non-overlapping or the probe is not active.

Loss of access tokens

The most common case of lost access tokens would be at startup since units initially are without resources and then acquires them by probing. Access tokens can also be lost by message losses on the control channel because of congestion, buffer overflows etc.

Doubly allocated tokens

This is the most serious case of faults in DRMP since it implies that two or more units have write access to the same resources at the same time. In this case, integrity of the data transported in the channels using these resources is violated. DRMP has been designed with the main objective of never allowing this to happen. It has been shown however that if a number of factors work together, it might nevertheless occur. Therefore, the kill mechanism exists to allow recovery from those unusual

situations. An important note to make here is that even if the owner has the resource in use locally it must probe periodically anyway, since someone else may wrongly be using that same resource.

Allocation domain

An allocation domain is a set of interfaces connected in sequence with each other. The allocation domain type can be either of point-to-point, bus or ring. In the bus cases, the last interface is defined not to be part of the allocation domain. Its transmitter is not used and hence it needs no outgoing resource. The allocation domain interpretation of a DLSP topology is also what is received via a DLC message. See figures 1a - d for examples, The figures 1a - d show: a five-node bus, ring, two-node bus and point-to-point topologies respectively.

Short circuit

Two or several interfaces may belong to the same node. This is sometimes referred to as "short-circuited" interfaces. Figure 2 shows an example of this. This is handled the same way as if C1 and C2 would have been on different nodes, with the exception that C1 and C2 communicate internally in the node and that channels originating in C1 may not pass beyond C2 and v.v.

A short-circuit is two interfaces that both belong to the same node and to the same allocation domain. In this case the ports C1 and C2 "speak" to each other using the ordinary DRMP messages, via a node-local control channel.

Probe

The probe is the mechanism responsible for detecting that there is a resource token missing and if so, recreate that token locally. Probing only takes place for slots that we consider owned. The probe always asks all the other nodes in the allocation domain if they use the resource. If not, then the resource token is recreated. The probe is the only mechanism that will recreate tokens (other mechanisms are only responsible for transferring tokens around). To avoid the risk of ownership overlaps and

thus the risk of double bookings, the probe is turned off during ownership changes.

Figure 3 shows a typical probe session. It is a probe that just checks that a borrower still has the slot. Probing is done because of two reasons:

- At bootup to get the resources initially, remember that all nodes must ask around for its resources since they may be borrowed out from a previous owner.
- Tokens can get lost when transmitted from one node to another.
- The node has just started. It always starts with no resources but with an ownership range assigned. This means it also has a responsibility to probe its resource tokens at a regular interval. At bootup, the probe is intensified, (sometimes called the turbo probe). The turbo strategy is to make at least one successful probing for each resource unit and when that is done go to probing at lower rate (and thus spend less CPU). The turbo is also re triggered whenever the topology of the specific allocation domain changes (due to fiber or node failures).
- Messages get lost on networks. The reasons are many, congestion in buffers or bit errors on the transmission media. The assumption is however that these occasions are rare and thus we settle for a relatively slow mechanism for the detection of lost resources.

<u>Borrowing and lending of token(s)</u>

Tokens can be borrowed from other nodes in the allocation domain. In figure 4 we see a typical borrow session. Interface A1 borrows from B1. After some time the channel is torn down and the borrowed resource is no longer needed and is thus returned.

<u>Ownership of token(s)</u>

The ownership of tokens is a process orthogonal to the access right. Changing ownership does not necessarily change the access right and vice versa. The intention of the distributed ownership defined in this protocol is to try to have the resources where they are needed, since borrowing and lending yields a

higher cost resource-wise than a simple local allocation at the local interface that needed the resource.

<u>Access to token(s)</u>

The access to tokens means that we either are already using the resource or have the right to do so, for instance by holding it in the local free list or using it for an active channel.

<u>Gather</u>

The message sent out from an interface that is "unhappy" with the current ownership distribution, or rather, the last received Sync message, (see Figure 6 and forth for examples), did not give the same limit for this interface as what we have locally.

<u>Sync</u>

A message initiated from the master node only or a node which thinks it is master, aiming at having all interfaces have the same idea of what resources they own.

<u>Bootup</u>

The process taking place when the power is switched on.

<u>Master interface</u>

The master interface is defined as the node having the lowest mac address. At any given time there is no guarantee that the nodes have consistent topology info.

<u>Transitional master interface</u>

This is the name of an interface, which becomes the master for a short period, since it has not yet got the correct topology information. Other interfaces may have other info on the topology and the mastership. Since the full allocation domain information is contained in the sync message, it is quite easy for an interface to determine that a sync message should be ignored.

<u>Probe</u>

A mechanism aiming at doing consistency checks for the allocation domain. It is also responsible for resolving resource conflicts and re-creating tokens that have been lost or, at boot time, do an initial create of the resources.

Topology

A Topology is a set of interfaces given to us as an ordered list from DLSP. A topology is either ring or bus and the last terminating interface is included in the topology.

Fairness algorithm

This is the algorithm that is used by the master of the dynown system to calculate the total amount of slots that each node should have. Please note that it is the policing parameters that are sent from each node which are distributed to each node and the calculation of the fairness takes place at each node, not only at the master. Therefore it is important that the fairness algorithm is defined in a precise manner.

Range change

This is an incoming event to the executive dynown telling it how many slots a specific interface would like to have. If the requests for the total amount of slots exceed the total available range, a fairness algorithm is used for this calculation.

Double booking

Double booking is said to have occurred when two or more interfaces believe they have access right to a certain resource unit or set thereof. There are two cases of this, disastrous and potentially disastrous. The disaster is defined as at least one slot being used for a channel on at least two nodes when the scope of the slot(s) are overlapping.

Intrinsic message

An intrinsic message, as opposed to an incoming or outgoing message, is a message that goes from one instance of a module on one interface to another instance of the same module in an other interface. The probe is a good example of an intrinsic message, since it is only the respective DRMP instances in two or more nodes that talk to each other. See figure 3 and 4 for examples of DRMP-intrinsic message passing.

Quark

This is the smallest resource unit available. It is defined to be one slot "high" and one physical link "wide". The quark machine

defines resource management with the assertion that any implementation will be an aggregation of several of these quarks in bulk for various optimizations. When uncertain how to handle a special case or event in the implementation, refer to the behavior of the quark machine.

Mine

An event telling the quark machine it now owns its quark.

Not mine

An event telling the quark machine it does no longer own its quark.

Alloc

Telling the quark machine its resource is used for a channel.

Dealloc

Telling the quark machine its resource is no longer used for a channel.

Fragmentation

This is pretty much the same type of fragmentation that occur in computer disks and memories. Several small channels are first allocated consecutively. Deallocation is not done consecutively; hence we end up with fragments of slots that require more computation and memory.

Worst case fragmentation

Sometimes the fragmentation of the resources has to be limited. The reasons may be many. The worst case of fragmentation is the case where the busy/free resource map resembles a checkerboard. It is used to calculate maximum message sizes though no formal proof exists that it generates the largest possible messages.

Bitrate

This is a measure of the capacity of a channel or a link. It is measured in slots. A slot is defined to be 512 kb/s. For example: 200 slots are about 100Mb/s of bitrate.

DETAILED PROTOCOL DESCRIPTION

The Quark machine

The Quark machine is a model in which we assume we only have one resource unit moving around in the system. The idea is that this simplifies initial modeling and verification. It is also assumed that the generalization to larger resource units is straightforward since they can be handled in blocks.

Simplified graphical diagram

Here is the graphical description. Please note that only the "normal" transitions are shown in this diagram. For all events on all states, see the below tables.

Events

- QReq (Quark Request) - This message is a request for a token it does not in it self change the state of the global system.
- QT (Quark Transfer) - This means that a token is being sent from one interface in the allocation domain to another. This token is also said to belong to a session. That is, it is aimed at a specific channel in the borrowing node. There are two possibilities: The resource message reaches the destination and the two state-machines changes state or the resource message gets lost for some reason. Only the state machine where the message leaves changes places.
- Qret (Quark Return) - Similar to QT, but this is just returned to another interface in the allocation domain and left in the free resource pool at the other node.
- Mine - This represents a change of ownership such that the local interface that gets the message now owns the resource.
- NotMine - This represents a change of ownership such that the interface that gets the message is no longer responsible for managing the resource.
- Alloc - This is a local request that should only occur on a Free resource. Most implementations will probably not be able to do anything else, since the resources are taken from a pool of available resources. The resource is allocated and moves to state Busy.
- Dealloc - A local resource is returned to the pool of Free resources.

- Probe - This message is used to ask other interfaces if they are using a certain resource.
- PrRpy - This message is the reply for the Probe, which tells us the state of the resource at that specific interface.
- Timeout - Currently, this only happens in the Probe state. (Remember that sending a QReq does not change the state of the Quark machine at the node that initiates the message).

<u>States</u>

- Free - Resource is free to use.
- Lent - Resource is used by someone else.
- Gone - We don't care where the resource is since we don't own it and don't use it.
- Borrowed - We don't own the resource, but we are using it.
- Busy - The resource is ours and we are using it.
- Probing - The resource is undergoing an examination of whether someone is using it or not.

<u>State transition tables</u>

This is all the states and events possible for the quark machine. The side effect "WARN" tells us that something has happened that "shouldn't". This means either one of two things:

- The event has become "impossible" due to implementation choices.
- The event is "illegal" in that it changes the system to a possibly dangerous, inconsistent or unknown state.

<u>Free</u>

The free state represents the fact that the resource is known to be available for use (either remote or local), that is, we *know* that no one else in the system is using the resource and that we are not using it ourselves.

Event	Condition	NextState	Action
QReq		Lent	send QT
QT		Free	WARN
QRet		Free	WARN
Mine		Free	
Not Mine		Gone	
Alloc		Busy	
Dealloc		Free	WARN
Probe		Free	
PrRpy		Free	
Kill		Lent	
time-out		--	

Table 1: Transition table for the Free state.

<u>Busy</u>

This state represents the fact that we know we are using the resource locally.

Event	Condition	NextState	Action
QReq		Busy	
QT		Busy	WARN
QRet		Busy	WARN
Mine		Busy	
NotMine		Borrowed	
Alloc		Busy	
Dealloc		Free	WARN
Probe		Busy	
PrRpy		Busy	
Kill		Busy	send Dcp Remove
time-out		--	

Table 2: Transition table for the Busy state.

<u>Lent</u>

This state represents the belief that the resource is in use somewhere else in the system and not at our node. The reason this is a belief is because this state represents the state of the system the last time we looked. Remember that in a distributed system things may have changed recently in other units without us knowing. This state is also the start state for resources

owned by the local node. At start we assume that someone is using the resource until we have asked everyone concerned.

Event	Condition	NextState	Action
QReq		Lent	
QT		Lent	WARN
QRet		Free	
Mine		Lent	
NotMine		Gone	
Alloc		Lent	WARN
Dealloc		Lent	WARN
Probe		Lent	
PrRpy		Lent	
Kill		Lent	
time-out		Probing	Send Probe's

Table 3: Transition table for the Lent state.

5 Gone

This represents the fact that we do not consider the resource to be ours and that we do not have it borrowed right now.

Event	Condition	NextState	Action
QReq		Gone	
QT		Borrowed	
QRet		Gone	
Mine		Lent	
NotMine		Gone	
Alloc		Gone	WARN
Dealloc		Gone	WARN
Probe		Gone	send PrRpy (Gone)
PrRpy		Gone	
Kill		Gone	
time-out		--	

Table 4: Transition table for the Gone state.

Borrowed
This means we have borrowed in a resource that someone else owns and that it is currently in use at our node.

Event	Condition	NextState	Action
QReq		Borrowed	
QT		Borrowed	WARN
QRet		Borrowed	WARN
Mine		Busy	
NotMine		Borrowed	
Alloc		Borrowed	WARN
Dealloc		Gone	send QRet
Probe		Borrowed	send PrRpy (Borrowed)
PrRpy		Borrowed	
Kill		Borrowed	send Dcp Remove
time-out		—	

Table 5: Transition table for the Borrowed state

Probing
This means that we have decided to ask for the resource but we have not yet obtained answers from all or any of the other nodes involved.

Event	Condition	NextState	Action
QReq		Probing	
QT		Probing	WARN
QRet		Free	
Mine		Probing	
NotMine		Gone	
Alloc		Probing	WARN
Dealloc		Probing	WARN
Probe		Probing	
PrRpy	Not Last PrRpy	Probing	p[i]=state
Kill		Probing	
time-out		Lent	
PrRpy	Last && InUse(p, state)	Lent	p[i]=state
PrRpy	Last && UnUsed(p, state)	Free	p[i]=state

Table 6: Transition table for the Probing state.

The functions inUse and unUsed deserve a special explanation. When we apply these functions they depend on the sub condition that the last probe reply has just arrived and that p contains all but the last reply. Remember that the condition is checked *before* we take action.

- InUse - This function is defined as true if *any* of its arguments are true. If more than one of its arguments are true a protocol error has occurred. This should trigger the kill function. The unUsed function implies that *all* of p is false (i.e. unused).

- UnUsed - This function is true if none of its arguments are true. This tells us that a recreation of the resource should be done.

The Dynamic ownership machine

The DO machine is the system for ownership negotiations. The states Free, Lent, Busy and Probing mean we have ownership rights to a resource. The state Borrowed or Gone means someone else has. The aim of this protocol is to negotiate ownerships in such a way that they never overlap before turning on the probe.

Typical scenarios

The aim of the design of dynamic ownership distribution is to cater for all situations, common or uncommon. An attempt has been made to optimize for some main scenarios, such as minimizing the time for the bootup sequence. See the figures 6, 7 and . Either the master gets the DLC (from DLSP) last, or some other node (a slave) will not have gotten it. Remember that in a distributed system, it is impossible to tell what goes on in the other nodes.

Transition diagrams

This is the transition diagram for the executive part of the dynamic ownership distribution.

State types

The following types are used for state variables in the DO machine:

Type	Description
AD	A representation of an ordered set of mac addresses
Bool	see Abbreviations
Integer	An unsigned number at least 32 bits large.
SyncStates	A variable, which can take two values: Idle or Wait.
Interface	This is just a 48 bit mac
List<T>	Some kind of ordered template which can hold several of these variables.
Limit	The Limit is a structure with two attributes: An ownership limit and a starvation flag, Integer and Bool respectively.
StarvFlag	A flag that indicates if an interface is set to starvation mode or not (i.e. it does not accept any new channels to/from or over it).

Table 7: The types used in the DO machine

State variables

The following state variables exist for each local port on the local node:

Name	Type	Description
pending	AD	Holds the topology info pending before synchronization is done. Initial value should be the own interface if a range_change triggers or the DLC topology if DLC triggers.
sync State	Sync States	This holds the state we are currently in, Idle or Wait. Idle means we are happy and ready to issue and reply to probes, however we claim that all DRMP links should be Idle before the global isMaster call yields true. Wait means that we are waiting for a synchronization and thus does not probe on any of our interfaces. This is a security measure since an interface might suddenly become part of another link when we start to move fibers around.
me	Interface	This is the Interface identifier of the local interface, we should have this as one of the members in the active and pending lists. It is to be regarded as a constant for a given state machine.
limits	List <Limits>	This holds the pending ownership ranges. We compare this to the incoming Gather's and determine if a new sync should be done. This is done if the limits change. If we are not the master we only store the limits. The limits are to be assigned a default zero at first, stating that that specific node does not need resources. When that node receives the sync, it should send a Gather, with its real number, protesting in order to get resources.
mylim	Limit	This holds the ownership limit. The own limit that is set only by RangeChange message. Initially one element of zero for the local Unit, or, if this State machine instance is booted with range_change an initial element with the given limit.
stbit	StarvFlag	This is a flag, which is given to us via operator interface. Then it is sent with the gather message, the same way as with the limit. All implementations need not be able to do this, but all must handle the drop of resources when a starvation bit is received via the sync message.
stbits	List <StarvFlag>	This holds flags for all interfaces in the allocation domain, that have requested that any free resources crossing their physical link must be dropped, this is used in the sync message.

Table 8: The state variables on a per local interface basis.

Queries

Queries are typically implemented as synchronous function calls in local software.

Probe conditions

This message is a query from executive DRMP concerning whether or not it is allowed to probe or reply to probes right now. The criteria is as follows:

- For each known local interface, the state is currently set to idle. This means it is allowed to probe right now.
- One, several or all of the local interface states are set to Wait. This means it is not allowed to probe right now for *any* of the interfaces.

Implementations might gain from caching the value of this variable and only recalculate it on any event that alters the state of any interface and changes to/from Wait/Idle.

Incoming Message

Asynchronous events going into the DO machine.

DLC

Description

DLC(Data Link Change) comes originally from DLSP. When stored, it is to be regarded with the probeUnit view (one less interface if buss or ptp).

master(r)	state	Action	New State	Send
no	Idle	pending:=r, clear(oldlimits), limits[me]:=mylim	Wait	Gather(me, mylim)
no	Wait	pending:=r, clear(oldlimits), limits[me]:=mylim	Wait	
yes	Idle	pending:=r, clear(oldlimits), limits[me]:=mylim	Wait	Sync(me, r, limits)
yes	Wait	pending:=r, clear(oldlimits), limits[me]:=mylim	Wait	Sync(me, r, limits)

Table 9: Reception of DLC(Interface me, AD r)

Clear(oldlimits) means that we remove the entries in our limits-container for possible interfaces that have been removed from the topology, (as given by the DLC event), but we want to keep interfaces already there.

<u>RangeChange</u>

Description

This is done from the user interface of a specific node (resedit & friends). In the case of the master getting a range change we send the gather internally to "ourselves".

Action	Send
mylim:=l	Gather(me, mylim)

Table 10: Reception of RangeChange(Limit l)

If this message gets lost on the way to the master, we rely on periodic retransmission of Gather. If this interface is the master, the Gather is sent locally.

Intrinsic Messages

Intrinsic messages are message which talks from one instance of DO machine in one task to another in another task, (usually another code-executing unit).

Sync

Description

This is the sync message. It is threaded between the interfaces in order from Master to most downstream, then to most upstream and downstream back to the master. Different interfaces may have other info on the topology and the mastership. Since the full allocation domain information is contained in the sync message, it is quite easy for an interface to determine that a sync message should be ignored.

master (pending)	pending = r	mylim= l[src]	State	Action	New State	Send
X	no	X	Idle		Idle	
X	no	X	Wait		Wait	
no	yes	yes	Idle	limits:=l	Idle	DistOwn(l)Sync (me, r, l)
no	yes	yes	Wait	limits:=l	Idle	DistOwn(l)Sync (me, r, l)
no	yes	no	Idle		Idle	Gather(mylim)
no	yes	no	Wait		Wait	Gather(mylim)
yes	yes	yes	Idle	limits:=l	Idle	
yes	yes	yes	Wait	limits:=l	Idle	DistOwn(l)
yes	yes	no	Idle		Idle	Gather(mylim)
yes	yes	no	Wait		Wait	

Table 11: Reception of Sync(AD r, List<Limit> l)

Please note that the limits-parameter in the Sync message should be the set of limits given from the interfaces. The actual ownership distribution should be calculated at each node, this of course implies that this function must be the same everywhere. For each interface calculate the start and range, then pass it on to executive DRMP via the RangeChange event. This is because we want each interface to compare the Synced limits to its local mylim. If not happy, the slave sends a Gather with the proper

value. The reason for doing so is that several desired limit-vectors maps to a certain set of ownership distributions. For example, if there are 5 interfaces and 10 slots total, then setting the policing parameters of all nodes to the same value, will yield the same results (2,2,2,2,2) for the ownership limits, thus we couldn't know whether to send a Gather without our "new" limit or not.

The ring and bus figures 9a and 9b show the signal paths from the master interface (marked with a crown) C1. In the bus case the sync-messages goes C1-D1-A1-B1-C1, while in the ring case, we go C1-D1-A1-B1-C1. Please also note that even though interface e1 is present it is *not* part of the ownership negotiation.

Gather

Description

This is periodically sent to the master of a pending link. A 'yes' in the second column, implies that the range for this unit has already been transmitted.

master (pending)	limits[src] = l	State	Action	New State	Send
no	X	Wait		Wait	
no	X	Idle		Idle	
yes	no	Idle	limits[src]:=l	Wait	Sync(me, pending, limits)
yes	no	Wait	limits[src]:=l	Wait	Sync(me, pending, limits)
yes	yes	Wait		Wait	
yes	yes	Idle		Idle	

Table 12: Reception of Gather(Interface src, Limit l)

Outgoing Messages

DistOwn(List<Limit> l)

DistOwn can be issued several times, with the own-limits (l) changing for each l. The Probe must be off when sending this. If

any StarvFlag bits are set we must drop any free resources that we have accessright to on the scope of that interfaces physical link.

Time-outs

(Re) send Gather

The Gather is sent periodically so that even if a message is lost we will eventually get through to the master. This time-out should be in the order of several seconds, since it is transmitted on a regular basis.

master(pending)	Action
no	send Gather(me, mylim)
yes	

Table 13: Timeouts

Message formats

This describes the various messages in DRMP. Note that all fields are to be network order. Please also note that the actual formatting on the outgoing stream should be from right-to-left and top-to-bottom.

In some messages, we use the division sign "/". This is to be interpreted as an integer division, i.e. any fraction is immediately truncated by the operation itself. We also use modulo "%" which is interpreted as the remainder of an integer division.

Bits and pieces

This is repetitive parts of messages that are frequent in the actual messages below. They are not themselves to be regarded as complete messages.

Mac addresses field

The Mac address is always placed "to the right" in a DTM frame, with the layout below:

0	Mac address

Fields	Slot #	BitVec	Size	Description
Mac address	any	[47:0]	48	This is the field for the 48 bit mac address

Table 14: Mac address fields

Slot fragment

This pattern is common throughout the document, it is the way to transport tokens through the network.

amt n	start n	amt n+1	start n+1

Fields	Slot #	BitVec	Size	Description
amt n	any	[63:56]	8	Amount of these slots
start n	any	[55:32]	24	Start of this slot fragment
amt n+1	any	[31:24]	8	Amount of next slots
start n+1	any	[23:0]	24	Start of next slot fragment

Table 15: Slot fragment fields

Long slot fragment

If both amt fields have the value zero slot fragments have a slightly different meaning, this is shown below:

0	start	0	amount

Fields	Slot #	BitVec	Size	Description
start	any	[55:32]	24	Start of this slot fragment
amount	any	[23:0]	24	Amount of slots

Table 16: Long slot fragment fields

This slot fragment representation is more optimal for very large consecutive slot tokens.

Generic header

This is the generic header for DRMP. Please note that the destination below is the interface we are talking *about*. This is not necessarily the same as the interface used to receive the actual message on the network (Checkout a double bus for an

example of this, it becomes even more obvious when we have more than one interface in an Allocation domain on the same node).

Vr	Cmd	0	Destination mac Address

Fields	Slot #	BitVec	Size	Description
Vr	0	[63:61]	3	The version of the protocol (currently 0)
Cmd	0	[60:56]	5	The command of the protocol
Destination mac Address	0	[47:0]	48	Destination mac that we are talking about

Table 17: Generic header fields

<u>Statistical information</u>

These are messages that do not alter any state in the quark machine. They are merely used as information. In a distributed environment they only hold the right data if the system is in steady state. If we do employ dynamic ownership the message with code 2 is used. If we do not have dynamic ownership distribution, we use the code 3 and stuff the ownership start and range in the message as well. All implementations should handle both message types and if an announce with code 2 is received when employing static ownership, the fields Own Start and Own Range should be interpreted as being zero although they are not present in the actual message.

Resource Announce (Dynamic ownership)

This is used if the interfaces ownership distribution is dynamic

0	2	N	broadcast mac addr
0			Announce Source mac
0		br hw 1	Up stream port link layer address 1
br lw 1			Down stream port link layer address 1
0		br hw n	Up stream port link layer address n
br lw n			Down stream port link layer address n
0		br hw N	Up stream port link layer address N
br lw N			Down stream port link layer address N

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]||[47:0]	56	Header, Cmd=2
N	0	[55:48]	8	This is the number of fragments in this announce message
Announce Source mac	1	[47:0]	48	This is the mac of the announcing interface
br n	2n, 2n+1	2n[55:48]|| (2n+1)[63:49]	24	The bitrate announced for this scope.
Up stream port link layer address n	2n	2n[47:0]	48	The link layer to the interface which finishes off the scope of this announce fragment
Down stream port link layer address n	2n+1	(2n+1)[47:0]	48	The link layer address to the interface which starts of the scope of this announce fragment

Table 18: Resource Announce fields

PRV01-07-10

Resource Announce (Static ownership)

This is used if the interfaces ownership distribution is static

<table>
<tr><td>0</td><td>3</td><td>N</td><td colspan="3">broadcast mac addr</td></tr>
<tr><td colspan="3">0</td><td colspan="3">Announce Source mac</td></tr>
<tr><td colspan="2">0</td><td colspan="2">Own start</td><td>0</td><td>Own range</td></tr>
<tr><td colspan="2">0</td><td>br hw i</td><td colspan="3">Up stream port link layer address 1</td></tr>
<tr><td colspan="3">br lw 1</td><td colspan="3">Down stream port link layer address 1</td></tr>
<tr><td colspan="2">0</td><td>br hw n</td><td colspan="3">Up stream port link layer address n</td></tr>
<tr><td colspan="3">br lw n</td><td colspan="3">Down stream port link layer address n</td></tr>
<tr><td colspan="2">0</td><td>br hw N</td><td colspan="3">Up stream port link layer address N</td></tr>
<tr><td colspan="3">br lw N</td><td colspan="3">Down stream port link layer address N</td></tr>
</table>

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]||[47:0]	56	Header, Cmd=3
N	0	[55:48]	8	This is the number of fragments in this announce message
Announce Source mac	1	[47:0]	48	This is the mac of the announcing interface
Own Start	2	[47:32]	24	The start of the ownership of the announcer
Own Range	2	[23:0]	24	The range of the ownership of the announcer
br n	2n+1, 2n+2	(2n+1)[55:48]|| (2n+2)[63:49]	24	The bitrate announced for this scope.
Up stream port link layer address n	2n+1	(2n+1)[47:0]	48	The link layer to the interface which finishes off the scope of this announce fragment
Down stream port link layer address n	2n+2	(2n+2)[47:0]	48	The link layer address to the interface which starts of the scope of this announce fragment

Table 19: Resource Announce fields

<u>Access token passing</u>

These messages are used to request or change access rights for tokens.

<u>Resource Request</u>

This message is used to request a set of resource from one interface to another.

0	4	0	Destination mac Address	
0		br hw	Source port link layer address	
br lw			downstream port link layer address	
0				Session identifier

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=4
Source port link layer address	1	[47:0]	48	The address of the interface which requests to lend resources
requested br	1,2	1[55:48]|2[63:49]	24	The amount of bitrate requested
downstream port link layer address	2	[47:0]	48	The scope for the request, note that the start scope is implicit from the source interface address, since no node will need to borrow resources which does not start at its own scope
B	3	[31:31]	1	If set to one, we indicate that this node is capable of accepting Resource transfers that are multi part.
RtReq session identifier	3	[30:0]	31	An identifier which is to be sent back in the reply to the borrowing request, it is used by the receiver to distinguish between possibly many outstanding borrowing sessions, it needs only be unique for each allocation domain

Table 20: Resource request fields

<u>Resource Transfer</u>

This is used to transfer the access right from one interface to another. The source of the sent resource is not sent since it is not needed until we return the resources, but by then the owner may well have changed for all or part of the slots. A session identifier is used to map together the request with the reply in case more than one channel is being requested for borrowing at the same time. The session id is also what holds the information on what scope the resource transfer is valid for. In the example below we pad the last slot fragment to the right with 32 bits of zeros.

0	5	N	Destination mac Address	
amt 1	start slot 1	amt 2	start slot 2
amt n-1	start slot n-1	amt n	start slot n
amt N-1	start slot N-1	amt N	start slot N
0		Session identifier	

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]||[47:0]	56	Header, Cmd=5
Number of fragments	0	[55:48]	8	The number of fragments in this resource transfer
amt n	n/2+1	(n/2+1) [32*(n%2)+31: 32*(n%2)+24]	8	The slot amount for this fragment
start slot n	n/2+1	(n/2+1) [32*(n%2)+23: 32*(n%2)]	24	The start slot for this fragment
B	N/2+1	[31:31]	1	More to come Bit, this indicates that there will be more slots in a coming message.
RtReq session identifier	N/2+1	[30:0]	31	The session identifier provided by the borrower

Table 21: Resource Transfer fields.

<u>Resource Transfer Return</u>

This is used to return resources. Again, the owner does not care about nor does it keep records of who originally borrowed the resources. Note the scope fields needed in this message. Below the number of slot fragments is odd we stuff the last (rightmost) 32 bits with zeros.

<table>
<tr><td>0</td><td>6</td><td>N</td><td colspan="3">Destination mac Address</td></tr>
<tr><td colspan="3">0</td><td colspan="3">Up stream port link layer address</td></tr>
<tr><td colspan="3">0</td><td colspan="3">Down stream port link layer address</td></tr>
<tr><td colspan="2">amt 1</td><td colspan="2">start slot 1</td><td>amt 2</td><td>start slot 2</td></tr>
<tr><td colspan="2">amt n</td><td colspan="2">start slot n</td><td>amt n-1</td><td>start slot n-1</td></tr>
<tr><td colspan="2">amt N-1</td><td colspan="2">start slot N-1</td><td>amt N</td><td>start slot N</td></tr>
</table>

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=6
Number of fragments	0	[55:48]	8	Number of fragments in Resource Transfer Return
Up stream port link layer address	1	[47:0]	48	The start scope for the return of this resource (applies to all slot fragments)
Down stream port link layer address	2	[47:0]	48	The end scope for the return of this resource (applies to all slot fragments)
amt	(n-1)/2+3	((n-1)/2+3) [32*(n%2)+31: 32*(n%2)+24]	8	Slot amount of current fragment
start slot	(n-1)/2+3	((n-1)/2+3) [32*(n%2)+23: 32*(n%2)]	24	Slot start of current fragment

Table 22: Resource Transfer Return fields

Probe

The message responsible for sending out a question for one or more of the owned resources to check if they are still there. The ownership of the resources can be either statically defined in each node for each interface, or it can be negotiated via the dynamic ownership machine. In the dynamic case, it is not OK to send out probes at all times. Please refer to section 12.6.2.5.1 for the exact conditions in which we are allowed to send out probes. The receiver of this message must handle the case when unknown mac addresses come in to the system.

0	7	0	Destination mac Address		
0			Source port link layer address		
0			Upstream port link layer address		
0			Downstream port link layer address		
0			Probe sess id	amount	start slot

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]||[47:0]	56	Header, Cmd=7
Source port link layer address	1	[47:0]	48	Interface address of the interface probing for this set of resources
Upstream port link layer address	2	[47:0]	48	The start of the scope for which the probe is valid
Downstream port link layer address	3	[47:0]	48	The end of the scope for which the probe is valid
Probe sess id	4	[47:32]	16	An identifier mostly used for robustness against protocol faults such as multiple outstanding probes for same resource, needs to be unique per allocation domain
amount	4	[31:24]	8	Slot amount of current probed fragment
start slot	4	[23:0]	24	Start of current probed fragment

Table 23: Probe fields

Probe Reply

This is an answer to a request for resources. Note especially that only the downstream scope of the resources are used in the fields below, the upstream scope is implicit from the source of

this interface (remember that nodes are very unlikely to access borrowed resources that are upstream to them). Note that it is only the resources that are in used (borrowed) that are listed in this reply. During ownership transitions it is possible to get probes for slots that an interface owns itself. It is important to note that if we get a probe that contains queries for slots that are currently not in our ownership range, we must not answer that probe at all.

For the understanding of this message it is important to note that it is two-dimensional. That is, first we have a number of fragments, one for each active scope. Then we have a set of slot fragments valid for each scope. If in the figure 20 the number of slot fragments, M, is odd for one or any of the slot fragments, 32 bits of zero must be stuffed in the right hand 32 bits of that slot. It is not allowed to send out probe replies at all times, please refer to the earlier description for the exact conditions on when to reply to probes.

0	8	N	Destination mac Address		
probe sess id			Source port link layer address		
0		frags 1	downstream port link layer address 1		
Amt 1		Start 1		Amt 2	Start 2
Amt m		Start m		Amt M	Start M
0		frags 2	downstream port link layer address 2		
Amt 1		Start 1		Amt 2	Start 2
Amt m		Start m		Amt M	Start M
0		frags n	downstream port link layer address n		
Amt 1		Start 1		Amt 2	Start 2
Amt m		Start m		Amt M	Start M
0		frags N	downstream port link layer address N		
Amt 1		Start 1		Amt 2	Start 2
Amt m		Start m		Amt M	Start M

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]||[47:0]	56	Header, Cmd=8
N	0	[55:48]	8	Number of different scopes in probe reply
probe sess id	1	[63:48]	16	The id of this probe session, the receiver of this message should check so that this is not old
Source port link layer address	1	[47:0]	48	The source of the interface answering to this probe(remember that probe is broadcast so we need to distinguish the answers, which are unicast
frags n	f(n,1)	f(n,1)[55:48]	8	Number of fragments for a certain scope in a probe reply
downstream port link layer address n	f(n,1)	f(n,1)[47:0]	48	Down stream address of this probe (upstream is implicit, since no node will use borrowed slots upstream of its own location).
Amt n,m		f(n,m)[(m%2)*32+31:(m%2)*32+24]	8	The amount of slots in the fragment
Start n,m		f(n,m)[(m%2)*32+23:(m%2)*32]	24	The start of this slot fragment

Table 24: Probe reply fields.

The function f(n) in table 24 is defined by the formula below. It gives us the slot offset as a function of n and m.

This is the function f(n,m) which defines the slot index for various n´s and m´s. Note the variable M_n, which is defined by the number of slot fragments for each scope. We need to use

the sum formula since all the M_n's may have different values. Also, it $n \equiv 1$, the sum evaluates to zero.

$$f(n,m) = 2 + \frac{m+1}{2} + \sum_{1}^{n-1}\left(\frac{M_{n-1}}{2} + 1\right)$$

Ownership passing

These are the messages for negotiating ownerships. The starvation bit is a sort of "vertical" ownership information. When receiving a set starvation bit in a sync message, the interface receiving the valid sync message must drop all free resources it may have on that physical link. The interface put in starvation mode must not probe nor respond to probes when it is put in starvation mode via the operators interface.

Sync

This message is originated from the master of an allocation domain. The master waits a certain time for the message to reach back again. If it does not reach back, the master re-transmits.

0	9	N	Destination mac Address	
0			interface link layer address 1	
0			interface link layer address n	
0			interface link layer address N	
0	req own 1		0	req own 2
0	req own n-1		0	req own n
0	req own N-1		0	req own N

If the number of interfaces in the allocation domain is odd we pad the lower-leftmost 24 bits with zeros.

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=9
Number of interfaces in this allocation domain	0	[55:48]	8	This is the number of interfaces listed in this message, it is also equal to the number of interfaces this interface considered to have when the message was transmitted
S	n	48	1	This indicates that the specific interface does not accept channels in, out or through it
interface link layer address n	n	n[47:0]	48	An interface mac address in the list
req own n	(n-1)/2+3	(n-1)/2+3 [32*(n%2)+23 :32*(n%2)]	24	The requested ownership limit that this interface wanted last time the master got that information

Table 25: Sync fields

Gather

The gather is sent by any interface (including the master that then talks to itself) whenever a need to change the policing parameter arises.

0	10	0	Destination mac Address
0			interface link layer address
0			req. own

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]][47:0]	56	Header, Cmd=10
S	1	48	1	Starvation bit, this informs the master that we do not want any more new channels to/from or over this interface.
interface link layer address	1	[47:0]	48	The link layer address of the interface doing the request for a certain ownership range
req. own	2	[23:0]	24	The ownership range requested by this interface

Table 26: Gather fields

Kill

The owner of a resource issues this message after doing a probe, which detects double booking.

0	11	0	Destination mac Address
0			Kill slot

Fields	Slot #	BitVec	Size	Description
DRMP generic Header	0	[63:56]||[47:0]	56	Header, Cmd=11
Kill slot	1	[23:0]	24	The slot requested to be killed due to double booking

Table 27: Kill fields

Questions on system limits.

Worst case message sizes

A calculation on this needs to be done on a per implementation basis. The parameters defining the message sizes are the following:

- Fragmentation of resources
- Number of interfaces in an allocation domain.

It is out of the scope of this document to specify a maximum size of ctrl messages; implementations should look this up and adjust their messages accordingly.

CPU intensity and scalability

Unless one has unlimited CPU power it is advisable to implement some kind of CPU meter. Overload will occur in two typical situations:

- An interface, a node or a whole system is coming up and wants to quickly acquire all its resources. This will generate a lot of probe and probe reply messages.
- A node is having a lot of channel setup and/or tear down going through it, remember that although the network switching is solely done through hardware signaling still can be very costly and CPU overload may occur, especially in nodes centrally placed in the network.

In actual implementations it has been found that in many cases the best thing to do is to make each successful probe trigger another one. Then when all resources have been counted once, we go to a more slow recovery probe since message losses during normal operation are relatively rare in their occurrence.

Message reordering

The DCC protocol is assumed to maintain strict FIFO on the message in all normal cases of operation. Message losses are acceptable at all times, but should be kept to a minimum for performance reasons. Reordering is assumed to be very uncommon, but the kill message will handle the cases that still can occur.

Scalability of the protocol itself

The probing process can be quite costly for the nodes involved. However the cost is linear or near linear provided that there is an inexpensive way to filter incoming messages not directed to the node itself.

Consider the process of bootstrapping a system with N nodes and M slots.

Assume the largest possible message size of the system corresponds to k probe replies in the same message

Assume the ownerships (and thus also the obligation to probe for resources) is distributed even among the nodes.

- This discussion is valid for $N \geq 2$.
- M and k are independent of N.
- Each node sends out $\frac{M}{kN}$ probes.
- A probe consists of N-1 queries and N-1 replies.
- This gives us a total of $\frac{2M(N-1)}{kN}$ messages sent in the system.
- This can be rewritten as $\frac{2M}{k}(1-\frac{1}{N})$ and $\frac{1}{N}$ diminishes towards zero as N increases.
- This implies that the cost per node is equal to or smaller than $\frac{2M}{k}$ i.e. nearly constant as N grows.

Since the number of nodes increase when one employs a larger bypass chain the processing power also increases in the total system, but so does the number of processors sharing the work. This shows that the CPU cost of the system is constant if each node only gets the messages intended for it.

If filtering of messages not intended for a specific recipient is associated with a non-neglectable cost, an additional cost proportional to N is introduced. This is always the case for bitrate cost.

Probe optimizations

Here is a set of hints and descriptions on how to make probing more efficient.

Turbo probe

An optimization concept aiming at having a faster probe rate when we startup or have detected inconsistencies and a slower probe rate during normal operation. This is done to minimize the bootup time for a node. The node has just started. It always starts with no resources but with an ownership range assigned. Thus it also has a responsibility to probe its resource tokens at a regular interval. At bootup, the probe is intensified. The turbo strategy is to make at least one successful probing for each resource unit and when that is done go to probing at lower rate (and thus spend less CPU). The turbo is also re triggered whenever the topology of the specific allocation domain changes (due to fiber or node failures). When this initial probing has been completed we settle for a relatively slow mechanism for the detection of lost resources, since the probability of message losses in a DTM system is relatively low.

Re-triggered-on-reply probe

The turbo probe is the concept of sending probes at a faster rate when needed. The re-triggered-on-reply probe is a flow rate mechanism for that. Generally it is hard to make a smooth send-out of probes when using a timer. The idea of re-triggered-on-reply is to let the last probe reply message for a given resource or resource-set trigger the next transmission of a probe. Sending probes periodically on a timer seems like a good idea, but in a real implementation case it often results in packet bursts since the granularity of most operating systems clocks is quite coarse. Scheduling only a few hundred timers a second and therefore also a few hundred context switches would be

noticeable on the CPU meter even when not probing at all. However with a turnaround from the querying interface to the answering interfaces of around 1 ms we could easily achieve 1000 probes per second without buffer build-up problems. It has been argued that this will not work for longer networks, but it is unlikely that anyone will build anything else than point-to-point in a very long link (such as a transatlantic or nation-wide one). The advantage of the ptp is that since the allocation domains are local (one on each side), no messages need to be exchanged with the other side in order to retrieve the resources.

Naturally, all of the process steps, as well as any sub-sequence of steps, described above may be carried out by means of a computer program being directly loadable into the internal memory of a computer, which includes appropriate software for performing the necessary steps when the program is run on a computer. The computer program can likewise be recorded onto arbitrary kind of computer readable medium.

Generally, a range of the ownership to the resources for a particular interface are distributed according to the expression: $\lfloor (V_{max} \times P)/(\Sigma_{req}) \rfloor$;

where

V_{max} denotes a maximal number of resources in the system,

P denotes a number of resources requested by the particular interface, and

Σ_{req} denotes a sum of the number of resources requested by all interfaces.

However, in order to elucidate the practical consequences of the proposed method of allocating resources, three different calculation examples are shown below.

In a first example the system is presumed to have 2000 slots in total, i.e. V_{max} = 2000. Four interfaces A, B, C and D all request 2000 slots each, i.e. P = 2000. An equal range of

$\lfloor(2000 \times 2000)/8000\rfloor = 500$ is thus allocated to all the interfaces A, B, C and D.

In case (i) such amounts of resources are requested by the interfaces in the system that the quotient $P/(\Sigma_{req})$ results in a fractional number, and (ii) a total number of resources has been requested by the interfaces A, B, C and D, which is larger than the number of resources in the system, i.e. $/(\Sigma_{req}) > V_{max}$, the ownership to any surplus resources is allocated to the master interface according to the expression: $V_{max} - \Sigma_{alloc}$; where Σ_{alloc} denotes a sum of ranges already allocated to the interfaces, i.e. the master interface as well as one or more slave interfaces.

If however, less than the total number of resources in the system have been requested by the interfaces A, B, C and D, or exactly the total number of resources has been requested (i.e. $\Sigma_{req} \leq V_{max}$) no additional resources are allocated to the master interface.

In a second example the system is again presumed to have 2000 slots in total, i.e. $V_{max} = 2000$. A first interface A requests 1700 slots, i.e. $P_A = 1700$; a second interface B requests 2000 slots, i.e. $P_B = 2000$; a third interface C requests 500 slots, i.e. $P_C = 500$; and a fourth interface D requests 700 slots, i.e. $P_D = 700$. The third interface C is presumed to be the master interface. Now, the ownership ranges of the resources are distributed according to the following.

A: $\lfloor(2000 \times 1700)/4900\rfloor = 693$,

B: $\lfloor(2000 \times 2000)/4900\rfloor = 816$,

C: $\lfloor(2000 \times 500)/4900\rfloor = 204$, and

D: $\lfloor(2000 \times 700)/4900\rfloor = 285$.

$\Sigma_{alloc} = 693 + 816 + 204 + 285 = 1998$.

I.e. $V_{max} - \Sigma_{alloc} = 2000 - 1998 = 2$. This means that the master interface C is allocated the ownership of an additional 2 slots,

thus in total 204 + 2 = 206 slots.

Finally, we show an example where less than the total number of slots are requested by the interfaces in the system. Again 2000 slots are presumed to be available, i.e. $V_{max} = 2000$. A first interface A requests 200 slots, i.e. $P_A = 200$; a second interface B requests 2000 slots, i.e. $P_B = 300$; a third interface C (master) requests 100 slots, i.e. $P_C = 100$; and a fourth interface D requests 50 slots, i.e. $P_D = 50$. Since here, $\Sigma_{alloc} < V_{max}$ each interface is allocated the ownership to exactly as many resources as the respective interface has requested. Thus, the ownership ranges of the resources are distributed according to the following.

A: 200, B: 300, C: 100 and D: 50.

The remaining 1350 are unreserved and held available for use by any future interfaces in the system.

The actual resources owned by each interface is calculated by the respective interface on basis of the particular interface's topological position relative to the range of the ownership to the resources allocated to the interface. In practice, this means that a first interface in a sequence, for instance the first interface A in the example above, is allocated the ownership to the first resources in accordance with its range, i.e. $1 \rightarrow 200$. Correspondingly, a following interface, for instance the second interface B, is allocated the ownership to the next range of resources, i.e. $201 \rightarrow (201 + 300) = 201 \rightarrow 501$, a yet following interface, for instance the third interface C, is allocated the ownership to a following range of resources, i.e. $502 \rightarrow (502 + 100) = 502 \rightarrow 702$ and a last interface, for instance the fourth interface D, is allocated the ownership to the range of resources, i.e. $703 \rightarrow (703 + 50) = 703 \rightarrow 753$.

In general terms, this may be expressed as:

$$\Sigma tp_{lower} + 1 \rightarrow \Sigma tp_{lower} + R_i$$

where Σtp_{lower} denotes a sum of ranges for all interfaces having a lower topological position number than the particular interface, and R_i denotes the range for the particular interface.

The term "comprises/comprising" when used in this specification is taken to specify the presence of stated features, integers, steps or components. However, the term does not preclude the presence or addition of one or more additional features, integers, steps or components or groups thereof.

The invention is not restricted to the described embodiments in the figures, but may be varied freely within the scope of the claims.

Claims

1. A method of allocating resources in a synchronous time division multiplex communications system having a master interface communicating with one or more slave interfaces, and in which the resources between the interfaces are represented by time slots in a repeating frame structure, **characterized by** the steps of:

sending a link status message from an interface whenever the interface registers a change in the topology of the system,

sending a gather message from an interface whenever the interface requests a revision of a current ownership distribution of resources,

sending a sync message from the master interface as an indication of a current distribution of ownership with respect to the resources between the interfaces in the system, and

for each interface generating a distribution of the ownership to the resources on basis of the interface's topological position and a latest received sync message.

2. A method according to claim 1, **characterized by** sending the link status message to all interfaces in the system.

3. A method according to any one of the preceding claims, **characterized by** sending the gather message to the master interface.

4. A method according to any one of the preceding claims, **characterized by** sending the sync message to all interfaces in the system, the sync message including information pertaining to a number of resources requested by the respective interfaces in the system, and the sync message being updated until all interfaces refrain from initiating any further gather messages.

5. A method according to any one of the preceding claims, **characterized by** generating a distribution range of the ownership to the resources for a particular interface according to: $\lvert (V_{max} \times P)/(\Sigma_{req}) \rvert$;

where

V_{max} denotes a maximal number of resources in the system,

P denotes a number of resources requested by the particular interface, and

Σ_{req} denotes a sum of the number of resources requested by all interfaces.

6. A method according to claim 5, **characterized by** allocating an additional range of the ownership to the resources to the master interface according to:

$V_{max} - \Sigma_{alloc}$; if $\Sigma_{req} > V_{max}$, and

0; if $\Sigma_{req} \leq V_{max}$

where Σ_{alloc} denotes a sum of ranges allocated to the master interface and the at least one slave interface.

7. A method according to any one of the claims 5 or 6, **characterized by** allocating ownership to the resources for a particular interface with respect to the interface's topological position relative to the range of the ownership to the resources allocated to the interface.

8. A method according to claim 7, **characterized by** allocating ownership to the resources for a particular interface having a range: $\Sigma tp_{lower} + 1 \rightarrow \Sigma tp_{lower} + R_i$

where

Σtp_{lower} denotes a sum of ranges for all interfaces having a lower topological position number than the particular

interface, and

R_i denotes the range for the particular interface.

9. A computer program directly loadable into the internal memory of a computer, comprising software for performing the method according to of any one of the claims 1 – 8 when said program is run on the computer.

10. A computer readable medium, having a program recorded thereon, where the program is to make a computer perform the method according to of any one of the claims 1 – 8.

11. A communications system having transmission resources in the form of time slots in a repeating frame structure, in which the time slots are dynamically allocable, the system comprising at least two interfaces of which one is a master interface and at least one is a slave interface, **characterized in that** it comprises at least one node, which in turn includes one or more of the interfaces, the node being adapted to effecting the method of according to any one of the claims 1 – 8.

Abstract

The present invention relates to a dynamic allocation of resources in a synchronous time division multiplex communications system. The system is presumed to include a master interface, which communicates with one or more slave interfaces. Furthermore, the resources are represented by time slots in a repeating frame structure, such as in a DTM system. The invention involves sending a link status message from an interface whenever the interface registers a change in the topology of the system. A gather message is sent from a particular interface to the master interface whenever the interface in question requests a revision of a current ownership distribution of resources. The master interface sends a sync message as an indication of a current distribution of ownership with respect to the resources between the interfaces in the system. Each interface generates a distribution of the ownership to the resources on basis of the interface's topological position and a latest received sync message.

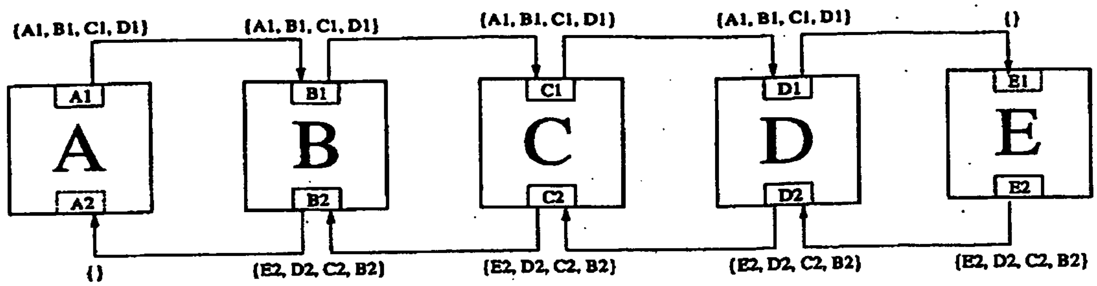


Fig. 1a

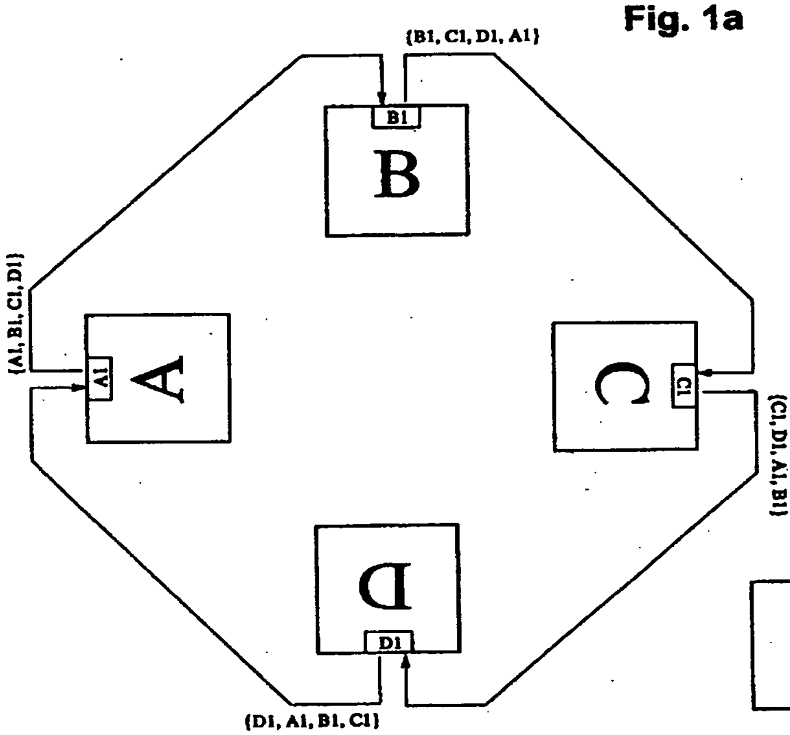


Fig. 1b

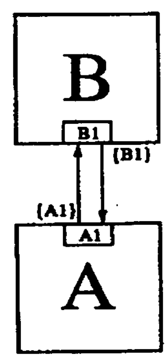


Fig. 1c

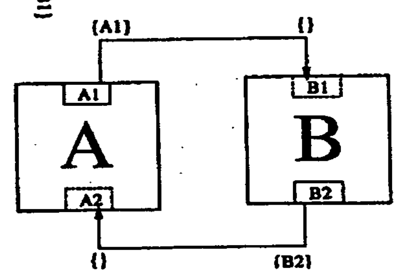
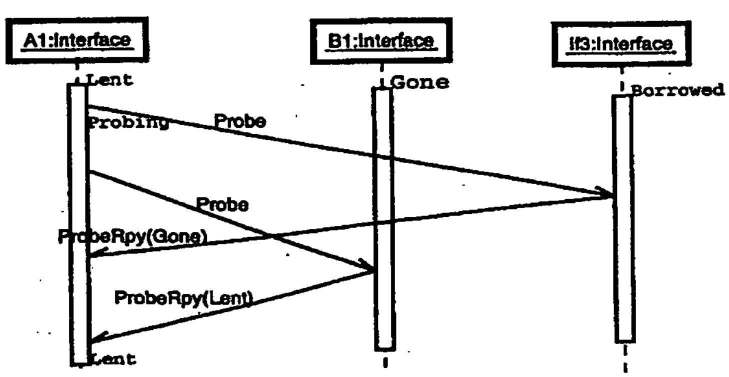


Fig. 1d



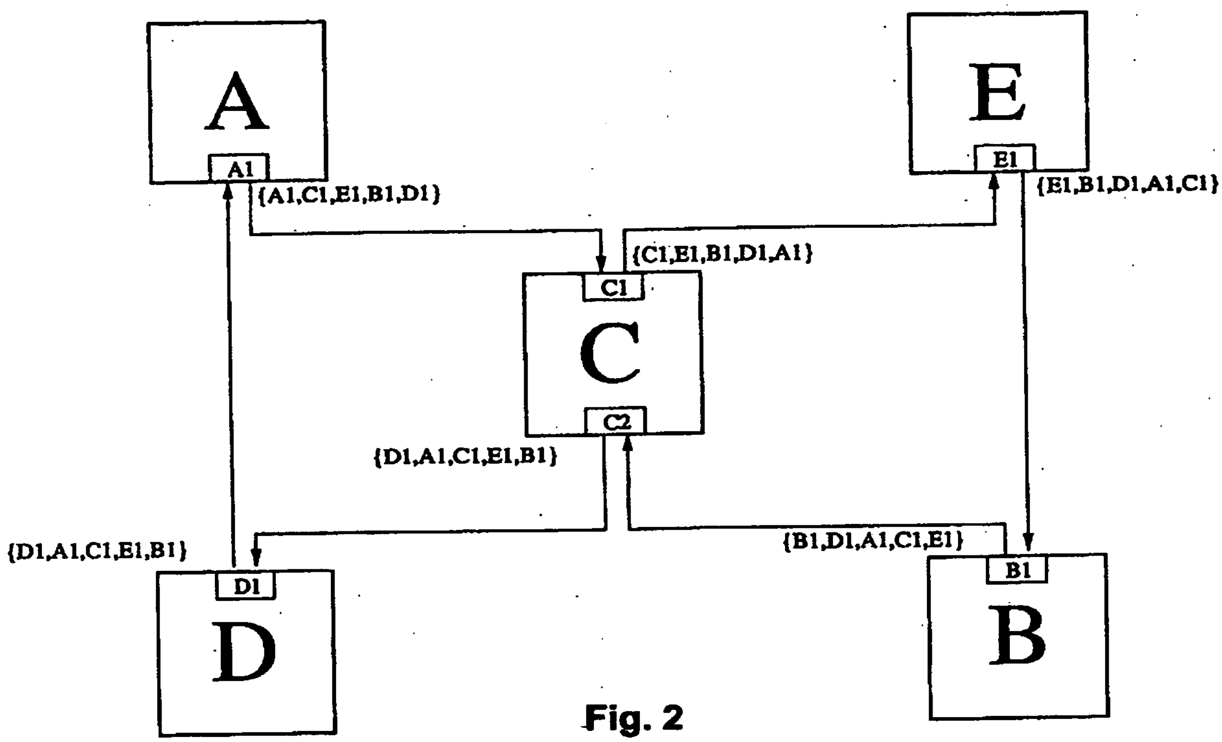

A
A1
{A1,C1,E1,B1,D1}
E
E1
{E1,B1,D1,A1,C1}
{C1,E1,B1,D1,A1}
C1
C
C2
{D1,A1,C1,E1,B1}
{D1,A1,C1,E1,B1}
{B1,D1,A1,C1,E1}
D1
D
B1
B

Fig. 2

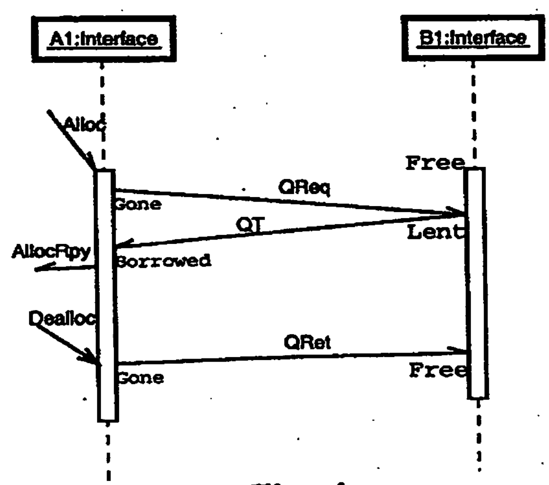

A1:Interface
B1:Interface
Alloc
Free
QReq
Gone
QT
Lent
AllocRpy
Borrowed
Dealloc
QRet
Gone
Free

Fig. 4

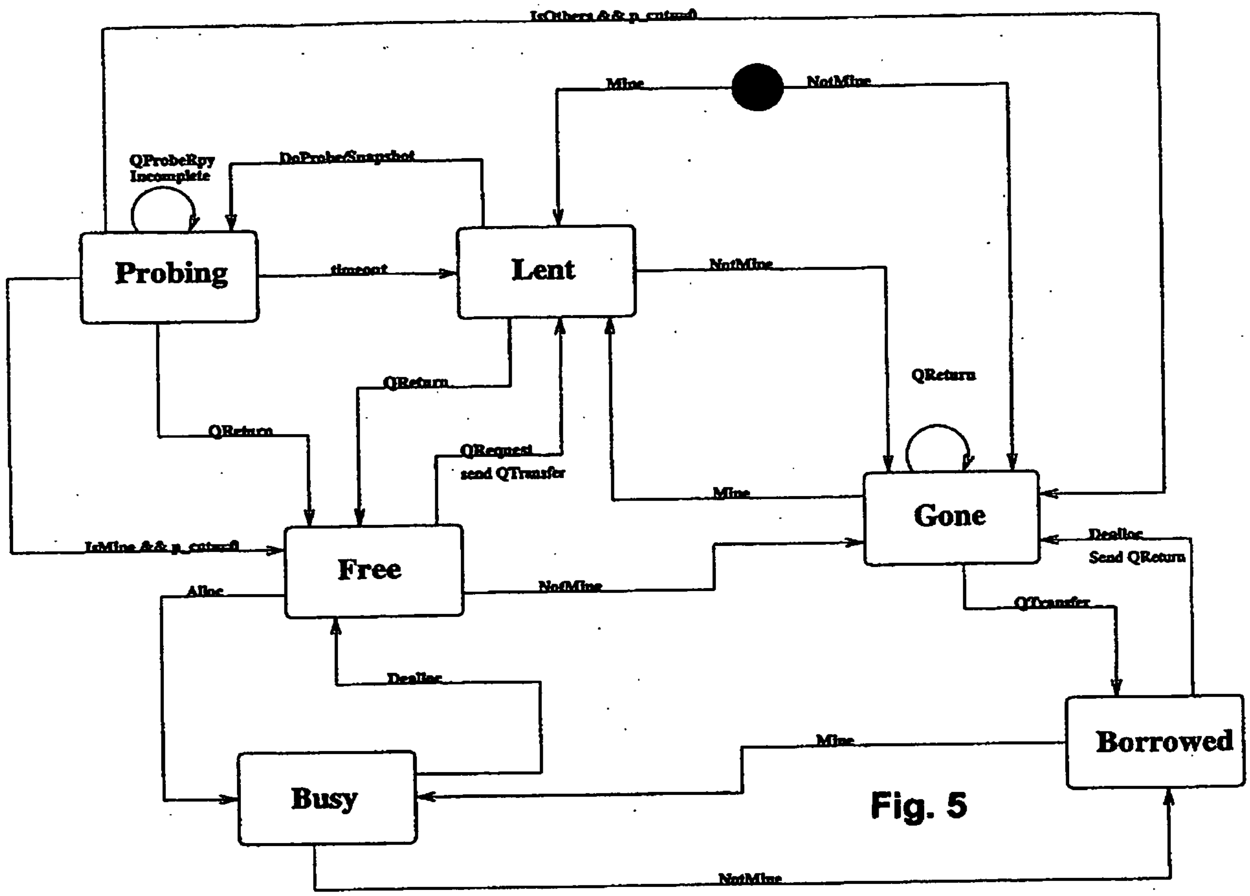
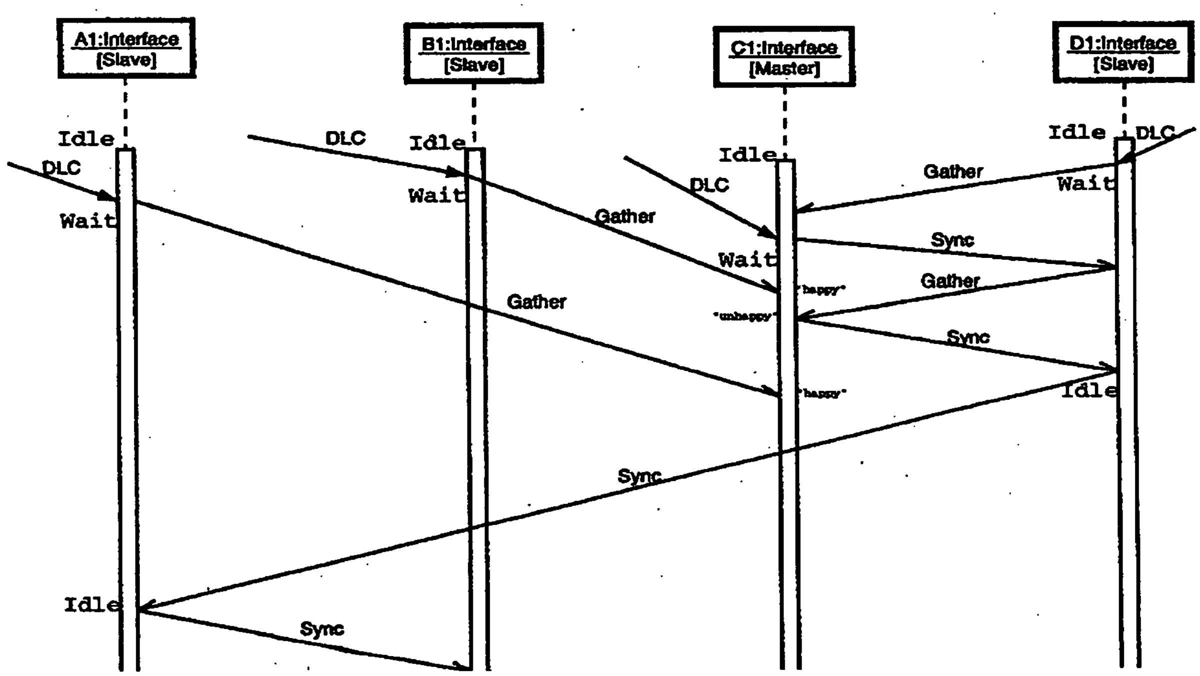

Probing
Lent
Gone
Free
Busy
Borrowed
Mine
NotMine
QProbeRpy Incomplete
DoProbe/Snapshot
timeout
QReturn
QRequest
send QTransfer
IsMine && p_cnt==0
IsOthers && p_cnt==0
Alloc
Dealloc
Dealloc
Send QReturn
QTransfer

Fig. 5


A1:Interface [Slave]
B1:Interface [Slave]
C1:Interface [Master]
D1:Interface [Slave]
Idle
Wait
DLC
Gather
Sync
"happy"
"unhappy"

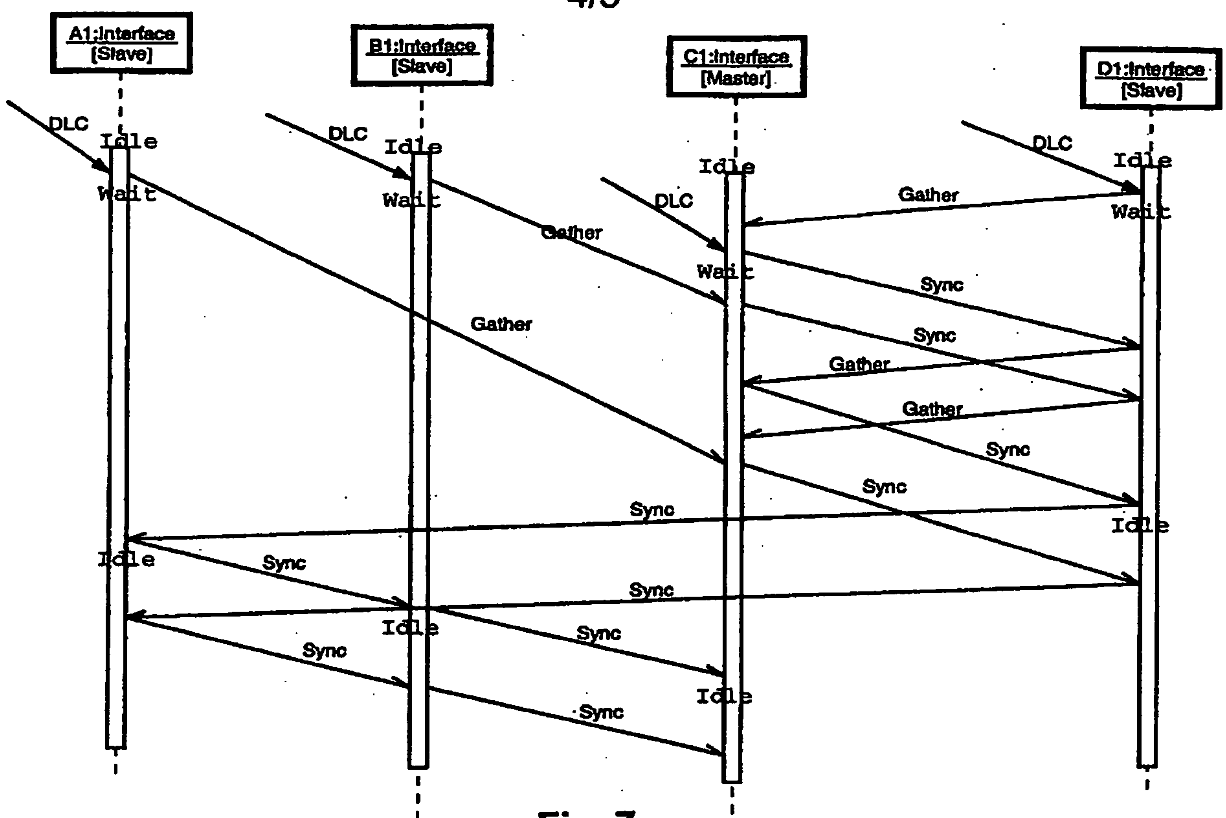
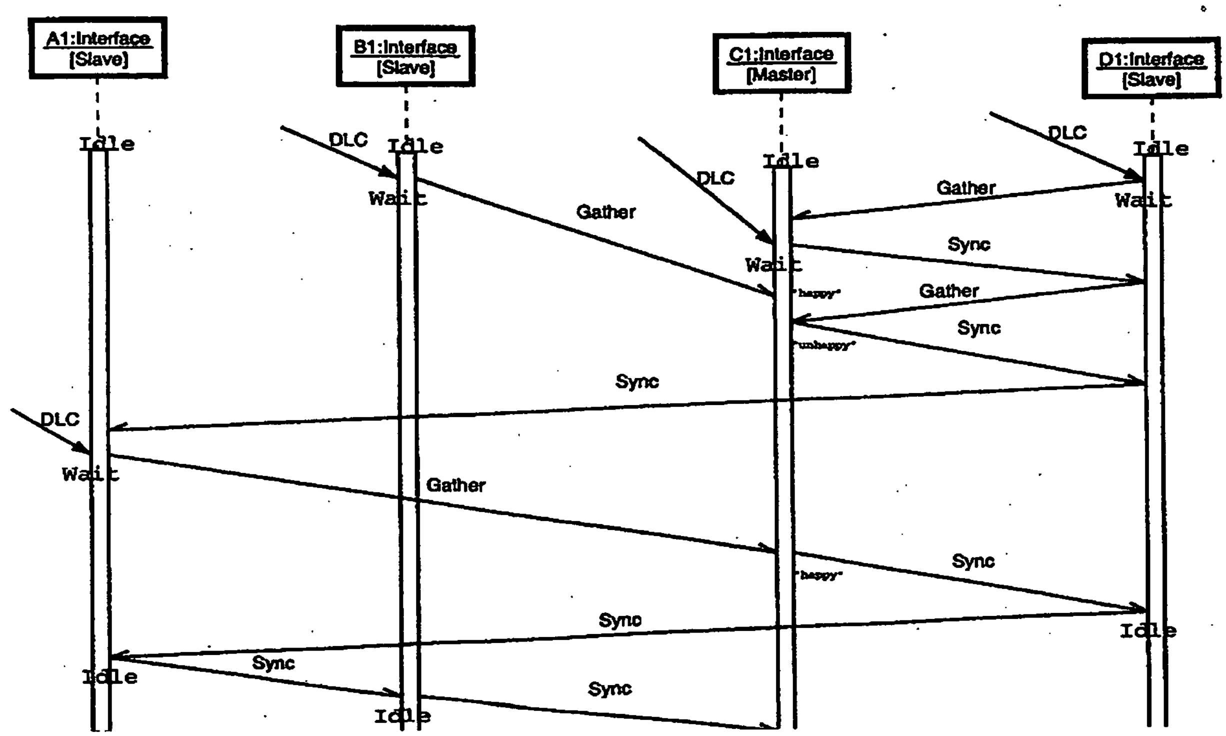

A1:Interface
[Slave]
B1:Interface
[Slave]
C1:Interface
[Master]
D1:Interface
[Slave]
DLC
Idle
Wait
Gather
Sync

Fig. 7


A1:Interface
[Slave]
B1:Interface
[Slave]
C1:Interface
[Master]
D1:Interface
[Slave]
DLC
Idle
Wait
Gather
Sync
"happy"
"unhappy"

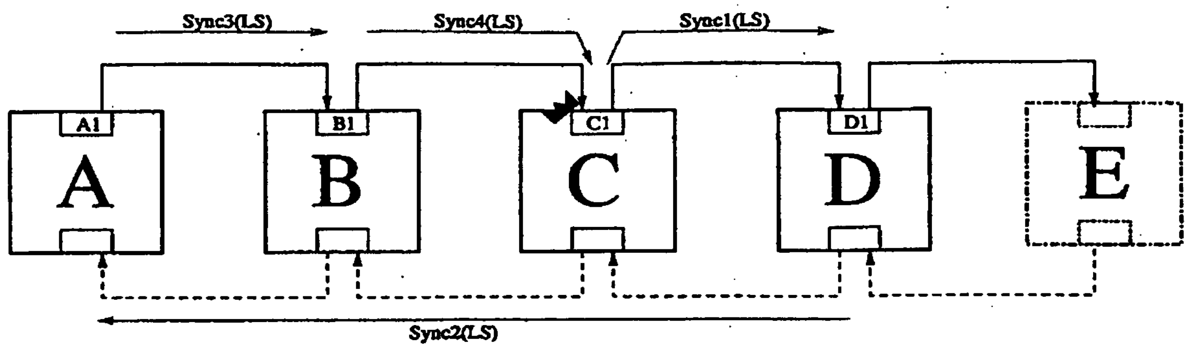

Sync3(LS)
Sync4(LS)
Sync1(LS)
A1
B1
C1
D1
A
B
C
D
E
Sync2(LS)

Fig. 9a

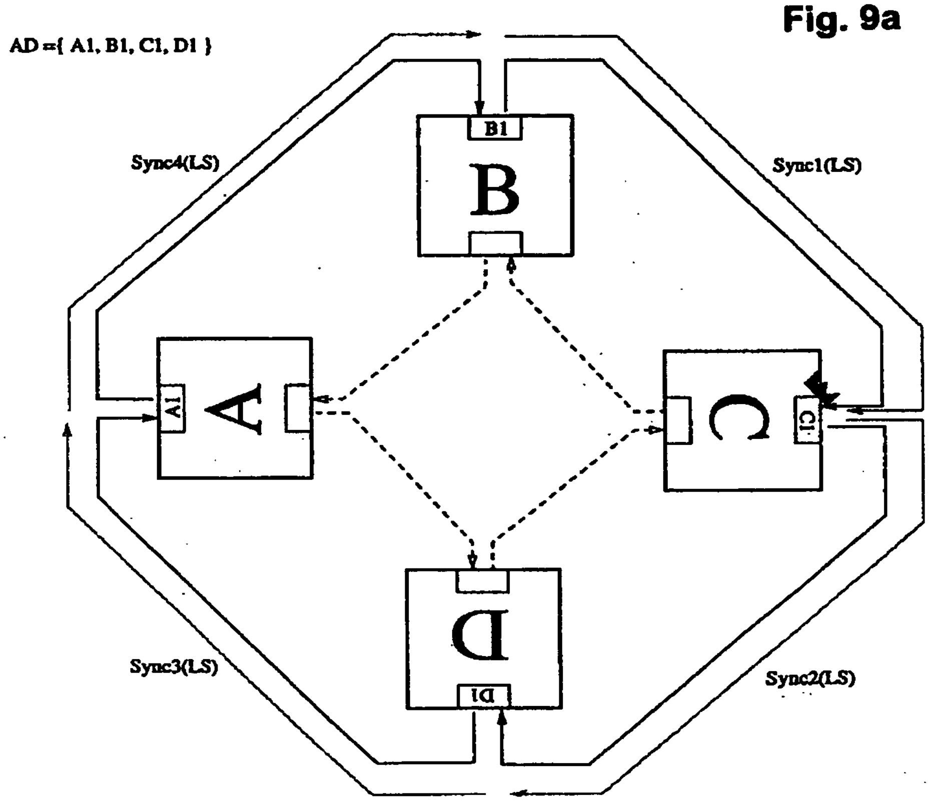

AD ={ A1, B1, C1, D1 }
Sync4(LS)
Sync1(LS)
B1
B
A1
A
C
C1
D
D1
Sync3(LS)
Sync2(LS)

Fig. 9b